Query and data caching in grid middleware

Laurent d'Orazio Cyril Labbé Claudia Roncancio
Fabrice Jouanot
Laboratoire d'Informatique de Grenoble
Adresse, France
firstname.lastname@imag.fr

Abstract

Caching is crucial to improve performances in many computing systems. This work proposes a caching approach for improving query evaluation. The proposed solution follows a semantic oriented approach and combines query and object caching. Separating query and objects provides high flexibility, by making possible to use several cooperations between caches and by supplying different query management tools. Our proposition has been experimented in a grid data management middleware and seems promising as showed by the results reported in this paper.

1 Introduction

Important work has been done in several areas to provide effective solutions for data sharing in large distributed systems. This paper describes a contribution to improve data transfer and processing in query evaluation. It has been tested in grid data management middleware. The main idea is related to the use of a variant of the view, a notion typical in DBMS. Our proposal relies on the use of partially pre-calculated queries handled by cache services. These semantic caches may be deployed according to the grid infrastructure and to user requirements. Queries may concern one or several sources distributed across the grid.

The semantic cache we propose is composed of a cache service, named *dual cache*, and a `query manager`. A *dual cache* is defined as a cooperation between a query cache and an object cache. The query cache keeps the calculated queries together with the identifiers of objects answering such queries. The object cache keeps accessed objects. Such a solution optimizes storage resource management, avoiding replication when objects are shared by several cached queries. In addition it saves computation, by making possible to cache a large number of query evaluations, even if corresponding objects do not all fit in the object cache. As a consequence, *dual cache* is particularly promising in a grid context, where resources (data sources and bandwidth) are shared among a large number of clients. The `query manager` can provide two main services: (1) *query capabilities* to allow local evaluation of queries on the object cache which can reduce server load, (2) *query matching* to compare received queries with pre-calculated ones in the query cache to enhance local query evaluation. Such a feature is particularly interesting in contexts where series of related queries will be issued in succession, with the results being at least partially overlapping.

This paper is organized as follows. Section 2 presents our proposition whereas section 3 presents a performance analysis in a middleware for data management on grids. Related work is described in section 4. Section 5 concludes this paper and gives research perspectives.

2 Dual cache and query management

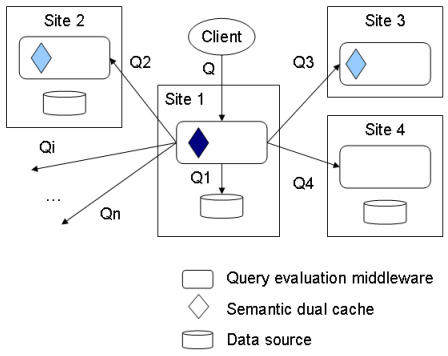


Figure 1. Evaluation on grids

Our proposal is designed to improve query evaluation over data sources distributed across a grid (figure 1). We propose a semantic cache [17] [12] solution integrating light weight query management capabilities (see figure 2). This proposal attempts to maximize advantages of semantic caching which are the reduction of both data transfers

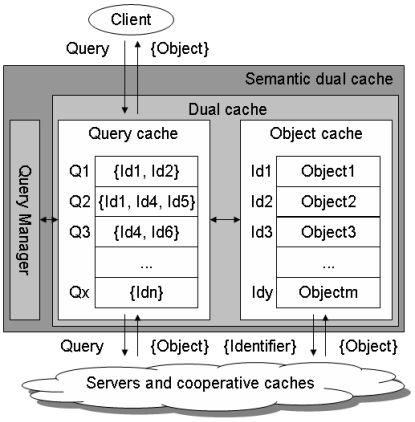


Figure 2. Semantic dual cache

and query computation. Our proposal, called *dual cache* clearly distinguishes these two goals by managing a couple composed of a query cache and an object cache. A query manager works together with the *dual cache*. Such a solution provides high flexibility to establish intra and inter site cooperation among the semantic caches deployed across the grid. In the following, section 2.1 presents *dual cache*, section 2.2 introduces query management aspects and section 2.3 discusses cooperation potentiality.

2.1 Dual cache

To explain our proposal, we consider a general query setting. Let `Qi` be a query to be evaluated on servers $\{$`Sj`$\}$. A standard processing of `Qi` implies sending queries to appropriate servers which send back partial answers. A global query manager composes the final answer. The answer to `Qi` is a set of objects having each an identifier, `ObjId`.

A *dual cache* is composed by a query cache and an object cache as illustrated in figure 2. *Dual caches* are intended to be deployed on user or proxy [21] sites[1]. They will rely on the resources of these machines to improve query evaluation. The query cache manages query results. Entries are identified by a query signature. The entry itself is the query answer stored as the set of identifiers of the relevant objects, `answer(Qi)=SetOf`$\{$`ObjIdk`$\}$. Objects themselves are in the related object cache, not in the query cache. When a new query is evaluated, answer retrieval implies loading the corresponding objects. So, a new query cache entry leads to object cache updates.

Dual cache uses partially pre-calculated queries which are close to the concept of a view in DBMS. Each entry `Qi` of the query cache plays the role of a pre-calculated query and `SetOf`$\{$`ObjIdk`$\}$ is its answer. As objects themselves are stored in an object cache which has no obligation to synchronize its content with the query cache, pre-calculated query or views may be full (every corresponding objects are stored in the cache) or partially (some objects are absent from the cache) materialized.

When a user query `Qj` is submitted to the *dual cache*, it may result in ”hits” or ”misses” in the query and the object caches. There is a query hit, if a `Qi` of the query cache can be used to answer `Qj` (see section 2.2). Otherwise there is a query miss. In this case, `Qj` is sent to the appropriate servers in the standard way or using the current cooperation policy (see section 2.3). If there is a query hit, object misses may or may not occur depending on the current state of the object cache.

The *dual cache* approach ensures some advantages in large distributed systems sharing large amounts of data. First, as cached objects may be shared by several cached queries, memory use is optimal. The independent management of query and object caches saves in many cases computation time and servers loads. Finally the communication between query cache and object cache and the configuration of each of them are very flexible: (1) the size chosen for each cache will determine whether data caching or computational caching is emphasized; (2) the level of cooperation between both caches may be as strong as desired according to the coherence between pre-calculated queries and cached objects; (3) the replacement strategy is decided for each cache.

It is to be noted that there are two main principles in a *dual cache*; firstly the query cache manages sets of identifiers, which are accessed by queries, secondly the object cache manages objects which are accessed by identifiers. This implies that servers or data sources must provide both access possibilities.

2.2 Light weight query manager

The `query manager` of the semantic cache isolates two distinct, but cooperative components to offer *query capabilities* and *query matching*. By *query matching*, we consider the semantic process of comparing a submitted query with the query cache content to deduce semantic overlap or semantic mismatch. By *query capabilities* we mean operators to locally evaluate queries on objects in the cache.

Query matching process analyzes queries to identify cache entries to be reused to answer a submitted query. When a query `Qj` is submitted, three types of hit may arise.
Exact hit: `Qj` is already pre-calculated in the cache. This is the best situation where the query was already submitted. In this case, the query cache contacts the object cache to retrieve objects in `answer(Qj)`. The object cache initiates cache misses resolution if necessary. The complete answer is returned through the query cache.
Extended hit: a pre-calculated query `Qi` subsumes `Qj`, the

[1] In special cases *dual cache* could be useful on server sites

results of `Qj` can be obtained from the answer to `Qi` i.e. the whole relevant object identifiers are present in `Qi` but some kind of filtering process is required, for example a selection or a projection. It can be achieved locally by the `query manager`.
Partial hit: `Qj` subsumes `Qi` or `Qi` overlaps without inclusion `Qj`. The answer of `Qi` is a part of the global answer of `Qj`. In this situation `Qj` is split in a *probe query*, which is the part know by the query cache and a *remainder query* corresponding to the missing part [12]. Objects in the answer to the probe query are retrieved as in the case of an exact hit. Remainder queries or miss resolution events can be solved by data servers or cooperative sites.
There is a *query miss* if `Qj` is totally disconnected from all pre-calculated `Qi`.

As *query matching* can be a very complex process, it is crucial to be able to judge when it is more effective to analyze query matching rather than consider a query as a miss. The complexity of this process is strongly related to querying capabilities of the `query manager`.

Querying capabilities are defined by operators (selections, projections, ordering, grouping, etc.) that can be evaluated by the *dual cache* on objects present in the object cache. As a matter of fact, when relevant object identifiers of a submitted `Qj` are present in pre-calculated query the *dual cache* uses its own *query capabilities* to process results. As it can be assumed that the cache will work with a small amount of data compared to the data managed by the server, it could be argued that the maximum of operator should be present in the cache. But, this would be interesting only if query capabilities are handled efficiently. One can ask, what are reliable situations for enabling these *query capabilities* and what are relevant operators to implement into the cache or to keep in the server side. As an example, if it is known that a special sorting algorithm fits better to an identified access pattern than the general one used by a server, then a sorting operator could be added to the cache *query capabilities*. Moreover, reducing the number of operators in the cache is also fruitful for the process of *query matching*.

Features built in a `query manager` provide high flexibility for deploying a cache architecture, enabling and configuring this component is the results of a trade off which is context dependent. It is important to take into account the complexity of typical query, resources allocated to the semantic cache, the server and network loads. It is also important to know the main purpose of the cache. If its final goal is to save servers resources then an enhanced `query manager` is required. All this knowledge has to be taken into account to choose the appropriate level of functionality for the `query manager`. The instantiation of a `query manager` may rely on works in the style of [11]. Our semantic cache approach provides a flexible architecture and modular components to enhance data access in various middleware. We are currently working to provide guide-line for well-configured semantic cache.

2.3 Cache cooperation

Resolution protocol [9] defines the process to retrieve data when a cache miss occurs. The choice of this protocol is very important in large distributed context. In fact, to avoid data sources to become bottleneck, one may prefer to make caches cooperate in order to balance the load. In that case, the usual strategy to resolve a cache miss is to contact other caches.

When the number of caches is really high, it is crucial to create relevant cooperations. Such cooperations may aim at reduce query evaluations or data transfer. However, these aspect are sometimes antagonist, making difficult for a classic cache to provide an efficient configuration.

As *dual cache*, clearly distinguishes evaluations and transfers, using two different caches, it enables new opportunities for making caches cooperate. In particular, it is possible to apply two distinct localities for the object and the query cache. While the former may aim at reducing data transfers by making cooperate object cache in a same area, the latter can use a semantic locality resulting in a cooperation between query caches having the same interests in order to avoid computations. Cooperative caching with *dual cache* is very flexible and can be customized, enabled or disabled according to system requirements. If query evaluation by servers is the bottleneck, cooperation between query caches is always relevant. However if retrieving objects from a data source is costly, sending an object request to siblings object cache is a good choice. In the case when a server is particularly efficient, it may be preferable to use it for resolution.

In addition to propose different groups for query and objects caches, it is also possible to choose distinct resolution protocols. While one may use a flooding resolution protocol as adaptations of *Internet Cache Protocol* (*ICP*) [27] or *Hyper Text Caching Protocol* (*HTCP*) [26], a catalogue based approach as in a *Cache Digests* [22] or *Summaries* [15] can be chosen for the other one.

3 Performance analysis

This section reports our experiences using semantic caching to improve querying in *Gedeon*, a middleware for data management in grids. Our main purpose is to compare existing semantic caching solutions [17] [10] to *dual cache* (*DC*).

3.1 Testbed configuration

Experimental data set and query server. Experiments have been done using *Swiss-Prot* [5], a biological database of protein sequences. It consists of a large ASCII file (750Mb) composed of about 210,000 sequence entries, each of then identified by an id. *Gedeon* middleware [25] provides a direct access to an entry through its id. It also provides query evaluation capabilities. Queries are composed of conjunctions and disjunctions of selection terms of the form `Attribute_name op value`. In the particular case of *Swiss-Prot*, `op` is often the `contain` operator and `value` is often a string. Evaluations result in a set of entries matching the query.

Caches under test. [17] presents a representative example of a semantic cache based on strict consistency between queries and objects without allowing replication of objects in the cache. This approach, used in web oriented databases [18], leads to an optimal use of memory space but, consistency between queries and objects in the cache has a cost.

On the other hand the proposal in [10] allows object replication in the cache. Such a replication prevents from focusing on consistency between queries and objects. When a region (corresponding to the answer of a query) is replaced, all corresponding objects are discarded. However a high level of replication and many redundant evaluations can occur. As seen before, *dual cache* avoids object replication in the cache and consistency management.

A Java and Fractal [6] version of *ACS* [14] has been used to instantiate the three caches under test: *dual cache* and caches proposed by [17] and [10]. Due to the server querying capabilities, the instantiated `query manager` for the three caches only cares of query containment using query signature [10]. *Query matching* only cares on detecting if a query is included in a cache entry and *query capabilities* are reduced to selection and conjunction operators. In this configuration partial hits never occur. In presented experiments, the cache size goes from 0.1 to 0.9 Gb and in all cases *dual cache* uses a size of 10 Mb for its query cache. When we studied the impact of workload, a 500Mb size caches has been used. This is large enough to be efficient but not so large to prevent the deployment of other applications. All caches under test use *LRU* replacement strategy.

Workload generation. Classical workloads used in benchmarks (TPC [1], proxy [4] and database Wisconsin[13], and Polygraph [2] for instance) do not consider semantic locality, whereas we consider it as an important behaviour for semantic caching. We use *Rx*, a synthetic semantic workload [20]. Queries correspond to progressive refinements. The first query is general and following ones are more and more precise and thus reduce the set of matching elements. In an *Rx* workload, *x* is the ratio of subsumed queries. For example, with *R50*, half of queries will be issued by constraining former queries. *R0* is equivalent to a uniform workload used in [10]. It is not the most suitable for semantic caching since it assumes queries do not present semantic locality. However, if a semantic cache is efficient in this context, it ensures this cache to be interesting for other access patterns. In presented experiments, workload is composed of queries corresponding to a single selection term, or to conjunctions of between two to four selection terms.

In presented experiments, the chosen workload goes from *R0* to *R90*. The impact of the cache size is studied with *R60* since it seems to be representative of our application context.

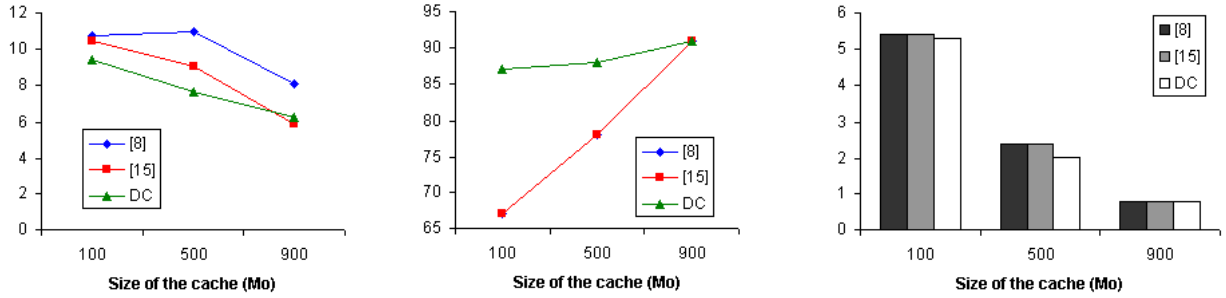
Performance metrics. One of the most important metrics to study is the mean response time which is strongly related to the hit ratio. But the server's load and the amount of data transferred from the servers to clients are also important metrics to be taken into account. As a matter of fact using a cache save servers and network resources. As a consequence selected performance metrics are: mean response time, hit ratio (exact and extended) and the amount of data transfered.

3.2 Single cache with a single server

The first experiments were done with a single server and a single client having the same characteristics (dual-Xeon 3Ghz, 2Gb memory, SCSI disk). Such performance analysis aims at understanding the comportment of the three caches with respect to their size and to semantic locality.

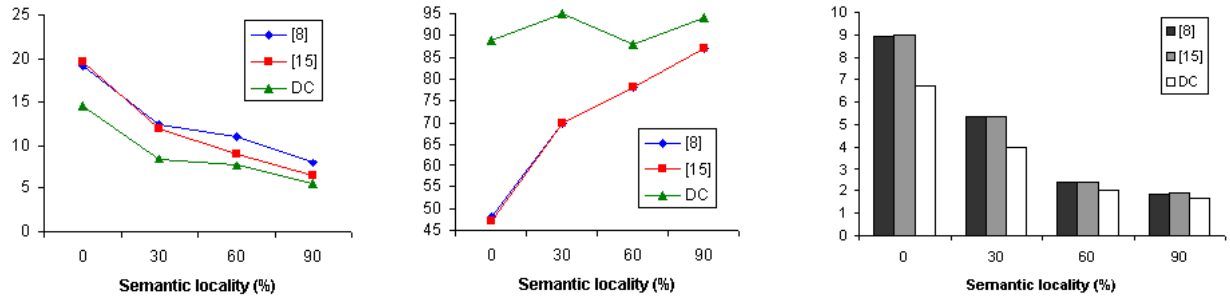
Figures 3 and 4 present the impact of the size of the cache and the semantic locality on the response time (3(a), 4(a)), the ratio of queries resolved by caches (3(b), 4(b)) and the amount of data transferred between the cache and the server (3(c), 4(c)). Note that for *dual cache*, results on figures 3(b) and 4(b) are about the query cache, and do not consider the objet cache. This later is taken into account in term of data transfer. Each workload correspond to 100 queries.

As expected, the efficiency of a cache increases with its size: the larger the cache, the higher the number of, exact and extended, hits. As a consequence the number of queries evaluated on the server decreases as well as data transfer leading to a shorter mean response time. The same positive behaviour is observed when semantic locality of submitted queries increases. It can be seen that for *dual cache*, size lightly influences query hits (3(b)), whereas semantic locality has no impact (figure 4(b)). In fact, even with a small size (10 Mb) the query cache is able to keep results of very large queries which often contain other queries. However,



(a) Mean response time (s) (b) Ratio of queries resolved by the cache (%) (c) Volume of data transfered (Go)

Figure 3. Impact of cache size (workload *R60*)



(a) Mean response time (s) (b) Ratio of queries resolved by the cache (%) (c) Volume of data transfered (Go)

Figure 4. Impact of semantic locality (0.5 Go cache)

for *dual cache* the number of object hits (not shown on this figure) is increasing with size and semantic locality.

3.3 Experiments in a grid: multi servers and multi clients

Experiments have been done on Grid5000 [7], the very large French platform for grid experiments. The data base has been partionned in three equivalent size files, managed by three clusters (Sophia Antipolis: bi-Opteron 2.2Ghz, 4Gb memory, SATA disk; Rennes: bi-Opteron 2Ghz, 2Gb memory, SATA disk; Toulouse: bi-Opteron 2.2Ghz , 2Gb memory, SCSI disk). Five nodes on each cluster are allocated to query evaluation. When a query is submitted, it is forwarded to the three clusters for a parallel evaluation. On each cluster, queries are randomly forwarded to one of the nodes used to evaluate queries. Ten clients placed on clusters (Sophia Antipolis, 4; Toulouse, 3; Rennes, 3) generate queries according to the *R60* workload and each of them uses its own 500Mb cache. The total number of submitted queries is 1000. For a single experiment all caches are of the same type, either [17], [10] or *dual cache*.

Table 1 describes the impact of semantic caching in a middleware for data management in grids. It can be noted that caches lead to a dramatic reduction in the number of communications with the servers since many queries result in exact or extended hits. As a consequence, the amount of data transferred from servers to clients is highly reduced. We can then conclude that a system with caches is more robust, as it saves server and network resources.

3.4 Interpretations

The experiments results on required server-client data transfers and cache hits are favourable to *dual cache*. It shows a more important reduction of the waiting time perceived by users during query evaluations. For instance, such a reduction is equal to 25% compared to [17] and [10] for a 0.5Gb cache with the *R0* workload. By separating queries from objects, *dual cache* and [17] enable optimal use of available storage resources. When all objects can be placed in the cache, [17] and *dual cache* present similar performances like it can be seen in the figure 3(a) with a 0.9Gb cache enabling to store the whole *Swiss-Prot*, whereas [10] is less efficient, because some query replacement occurs due to replication of objects shared by different queries. Unlike [17], *dual cache* allows to enter an answer in the query cache even if corresponding objects cannot be stored in the object cache, making possible to store myriad of queries. Such behaviour makes the *dual cache* particularly well suited if the size available for a cache is small. Our solution is particularly promising in this context be-

	Response time	Exact hits	Extented hits	Evaluation on servers	Transfered data
[10]	18.1 s	19.6 %	56.8 %	23.6 %	34.67 Go
[17]	16.5 s	26.8 %	49.9 %	23.3 %	35.02 Go
DC	15.6 s	47 %	39.8 %	13.2 %	29.50 Go

Table 1. Specific performance metrics in a grid context

cause retrieving object via their identifier is more efficient than evaluating corresponding queries. For example, in the one client one server experiment with the *R60* workload and a 500Mb cache, our solution is more efficient than [17] and [10] since it retrieves objects via their identifiers for 12% of the queries and makes the server evaluates 12% of them whereas this ratio is 22% for the two others. It has to be noticed that *dual cache* generates a better exact hits ratio, which explains its extended hit ratio is less important than [17] and [10].

4 Related work

Our experiments compared *dual cache* to proposals presented in [17] and [10] which are representative of the two main approaches used in semantic caching, those separating query and objects ([17] [18]) and those handling them together ([10] [12]).

However, there are other proposals. [16] caches parts of queries that can be reused for further evaluations. This solution considers semantic aspects but does not manage extended hits. [23] proposes a cache of views in a centralized system. Caching views is quite interesting, but as object caching is not considered such a solution may be of limited use in a distributed environment. On the same principles, [19] proposed caching views if they cannot be obtained using already materialized ones.

None of the mentioned works have been deployed in a grid. [3] and [8] do by adopting a mediation like approach[28]. *ICM* [3] focuses on the problem of network latency. It proposes to store data in distributed DB replicated across the grid. User SQL queries are submitted to the cache which decomposes them into sub-queries for local and remote domains, and builds afterwards the final results. [8] offers semantic cache functionalities by using hierarchical cache architecture. A kind of global cache federates grid node caches by using a global catalogue. A metadata catalogue helps to localize data in data sources. Our proposal is orthogonal to [3] and [8].

Dual cache is not the first system combining two caches. For example, [24] proposes a cache solution for parallel multiple sequence alignments. Such solution is composed of a cache for pair wise alignments and a second one to store multiple alignments. Pair wise entries are used to compute multiple ones. Contrary to *dual cache*, [24] is very context specific.

5 Conclusion

This paper presents a cache solution to improve querying in a grid context. The *dual cache* is based on the cooperation of a query cache and an object cache. This proposition has been implemented, tested and compared to other semantic cache propositions. Experiments have been done in a grid context using a data management middleware. Results proved the efficiency of our solution in this context. Our proposal saves computation time since it maximizes query caching and saves memory as the object cache avoids ”in cache object replication”. Cooperation between the two caches never introduces overhead related to consistency issues. Moreover this approach allows tuned configuration for each cache. This is particularly useful in a grid environment where caches may be deployed on heterogeneous sites.

Furthermore, thanks to a clear separation of calculus caching (query cache) and access caching (object cache) *dual cache* leads to new opportunities. First different kind of querying capabilities (filtering, grouping, ordering, etc.) may be used in the cache solution. Then, it is possible to resolve a cache miss in a different way for the query cache and the object cache, according to specific cooperation policies related to semantic or geographic locality.

Our solution seems well suited for contexts where shared data has low update rate. However, even in this context, consistency issues have to be developed. In addition, future work involves the study of various application contexts including warehouse oriented systems. We also plan more in-deep work on cooperation policies and replacement strategies. Finally, we are interested in context-aware caching strategies for developing self-adaptive and autonomous caches.

Acknowledgement: Thanks to S. Albrand for remarks on this paper, O.Valentin and Y. Denneulin for their contribution to this work, the *Gedeon* and *Hadas* teams for fruitful discussions, *Ministère délégué à l'Enseignement supérieur et à la Recherche* and *Institut National Polytechnique de Grenoble* for financial support.

References

[1] Tpc benchmarks. http://www.tpc.org/.

[2] Web polygraph, performance benchmark for web intermediaries. http://polygraph.ircache.net/.

[3] M. U. Ahmed, R. A. Zaheer, and M. A. Qadir. Intelligent cache management for data grid. In *Proc. of the WS on Grid computing and e-research*, pages 5–12, 2005.

[4] J. Almeida and P. Cao. Measuring proxy performance with the Wisconsin Proxy Benchmark. *Computer Networks and ISDN Systems*, 30(22–23):2179–2192, 1998.

[5] B. Boeckmann, A. Bairoch, R. Apweiler, M.-C. Blatter, A. Estreicher, E. Gasteiger, M. J. Martin, K. Michoud, C. O'Donovan, I. Phan, S. Pilbout, and M. Schneider. The swiss-prot protein knowledgebase and its supplement trembl in 2003. *Nucleic Acids Res*, 31(1):365–370, 2003.

[6] E. Bruneton, T. Coupaye, M. Leclerq, V. Quéma, and J.-B. Stefani. An Open Component model and its support in Java. In *Proc. of the Intl. symposium in Component based Software Engineering*, pages 7–22, 2004.

[7] F. Cappello. Grid'5000: A large scale, reconfigurable, controlable and monitorable grid platform. In *Proc. of the sixth IEEE/ACM Intl. WS on Grid Computing*, 2005.

[8] Y. Cardenas, J.-M. Pierson, and L. Brunie. Uniform Distributed Cache Service for Grid Computing. In *In Intl. WS on Database and Expert Systems Applications*, pages 351–355, 2005.

[9] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell. A hierarchical internet object cache. In *USENIX Annual Technical Conf.*, pages 153–164, 1996.

[10] B. Chidlovskii and U. M. Borghoff. Signature file methods for semantic query caching. In *Proc. of the 2nd European Conf. on Research and Advanced Technology for Digital Libraries*, pages 479–498, 1998.

[11] C. Collet and T.-T. Vu. Qbf: A query broker framework for adaptable query evaluation. In *Proc. of the 6th Intl. Conf. on Flexible Query Answering Systems*, pages 362–375, 2004.

[12] S. Dar, M. J. Franklin, B. T. Jonsson, D. Srivastava, and M. Tan. Semantic data caching and replacement. In *Proc. of the 22th Intl. Conf. on Very Large Data Bases*, pages 330–341, 1996.

[13] D. J. DeWitt. The wisconsin benchmark: Past, present, and future. In J. Gray, editor, *The Benchmark Handbook for Database and Transaction Systems (2nd Edition)*. Morgan Kaufmann, 1993.

[14] L. d'Orazio, F. Jouanot, C. Labbé, and C. Roncancio. Building adaptable cache services. In *Proc. of the 3rd Intl. WS on Middleware for grid computing*, pages 1–6, 2005.

[15] L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary cache: a scalable wide-area web cache sharing protocol. *IEEE/ACM Trans. Netw.*, 8(3):281–293, 2000.

[16] S. Finkelstein. Common expression analysis in db applications. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of data*, pages 235–245, 1982.

[17] A. M. Keller and J. Basu. A predicate-based caching scheme for client-server db architectures. *The VLDB Journal*, 5(1):035–047, 1996.

[18] D. Lee and W. W. Chu. Semantic caching via query matching for web sources. In *Proc. of the 8th Intl. Conf. on Information and knowledge management*, pages 77–85, 1999.

[19] K. C. K. Lee, H. V. Leong, and A. Si. Semantic query caching in a mobile environment. *SIGMOBILE Mob. Comput. Commun. Rev.*, 3(2):28–36, 1999.

[20] Q. Luo, J. F. Naughton, R. Krishnamurthy, P. Cao, and Y. Li. Active query caching for db web servers. In *Selected papers from the 3rd Intl. WS WebDB 2000 on The World Wide Web and DBs*, pages 92–104, 2001.

[21] A. Luotonen and K. Altis. World-wide web proxies. In *Selected papers of the 1st Conf. on World-Wide Web*, pages 147–154, 1994.

[22] A. Rousskov and D. Wessels. Cache digests. *Computer Networks and ISDN Systems*, 30(22-23):2155–2168, 1998.

[23] N. Roussopoulos. An incremental access method for viewcache: concept, algorithms, and cost analysis. *ACM Trans. DB Syst.*, 16(3):535–563, 1991.

[24] D. Trystram and J. Zola. Parallel multiple sequence alignment with decentralized cache support. In *Proc. of the 11th European Conf. on Parallel Computing*, pages 1217–1226, 2005.

[25] O. Vanlentin, F. Jouanot, L. d'Orazio, Y. Denneulin, C. Roncancio, C. Labbé, C. Blanchet, P. Sens, and C. Bonnard. Gedeon, un intergiciel pour grille de données. In *Conf. Française en Système d'Exploitation*, 2006.

[26] P. Vixie and D. Wessels. Rfc 2756: Hyper text caching protocol (htcp/0.0), 2000.

[27] D. Wessels and K. Claffy. ICP and the Squid Web cache. *IEEE Journal on Selected Areas in Communication*, 16(3):345–357, 1998.

[28] G. Wiederhold. Mediators in the architecture of future information systems. *Computer*, 25(3):38–49, 1992.